CLAIMS

I claim:

1. A computer program testing method for collecting internal test distribution information, and for indicating test diversity throughout source files written in the same or different programming languages; the method includes the steps:

   parsing and instrumenting the computer program to provide an instrumented computer program;

   executing the instrumented computer program to generate a test-distribution record and a path trace; and

   producing test diversity output using the test-distribution record and the path trace to indicate the internal conditional diversity, data diversity, and path diversity of the program.

2. A method of software testing as in claim 1 wherein the execute step involves:

   dynamically updating a test-distribution record of *true/false* frequency counts associated with each conditional expression and sub-expression in the program;

   dynamically updating a compact path trace consisting of the locations of the conditional expression in the code and their resulting Boolean values after they have been completely evaluated; and

   possibility of altering the normal control flow of the program by discarding the resulting Boolean values of the conditional expression, dynamically generating Boolean values, substituting these generated values for the discarded ones, and continuing execution with the generated values.

3. A method of software testing as in claim 1 further including the step of producing an audible report indicating conditional, data, and path diversity for the program under test.

4. A method as in claim 1 wherein the step of producing a diversity output for conditional diversity includes the steps of:

   calculating the conditional diversity for a conditional expression form a test-distribution record as a distance between the even distribution of *true* and *false* condition evaluations and the actual distribution for that expression;

calculating the sub-conditional diversity for a conditional expression form a test-distribution record as a distance between the even distribution of *true* and *false* sub-condition evaluations and the actual distribution for each sub condition in the expression;

calculating the average conditional diversity by averaging the conditional diversities for all the conditional expressions; and

calculating the average sub-conditional diversity by averaging the sub-conditional diversities for all the conditional expressions.

5. A method as in claim 4 wherein calculating the distance includes the steps of:

   calculating a distance for a two-way conditional statement as
   Average = (true hits + false hits)/2
   Distance = 1–((|Average–true hits| + |Average – false hits|)/(true hits + false hits); and

   calculating a distance for a multi-branching conditional statement as
   Average = total hits for all branches/number of branches
   Distance = (total hits for all branches – hits for branch)/Average.

6. A method as in claim 1 wherein the step of producing a diversity output for data diversity includes the steps of:

   calculating the conditional diversities for each of a set of multiple test suites;

   calculating the individual data diversity for each conditional expression as a percentage of test suites for which the conditional diversities for that conditional expression are distinct; and

   calculating the total data diversity as the average of individual data diversities.

7. A method as in claim 1 wherein the step of producing a diversity output includes the step of calculating the path diversity from a compact path trace as a percentage of conditional expressions for which, if the conditional expression evaluated to *true* on a path, it also evaluated to *false* on the same path, and vice versa.

8. A method for parsing computer software as in claim 1 by only parsing conditional statements and isolating the conditional expression and conditional sub-expressions in the conditional statement.

9. A method to instrument computer software as in claim 1 by inserting a function call around the conditional expression and conditional sub-expressions in a conditional statement to:

   dynamically evaluate conditional expressions and sub-expressions and immediately update the *true/false* counts based on the evaluation;

   to dynamically produce a compact path trace of conditional expression locations and their values; and

   to dynamically generate Boolean values, evaluate conditional expressions, discard the resulting Boolean value from the evaluation, and substitute the generated value for the discarded one.

10. A method as in claim 1 where the collection of distribution/trace data is cumulative, and where the permanent distribution records/traces are kept updated until they are initialized.

11. A method as in claim 1 where the permanent distribution records for different test runs are merged into a single permanent distribution record.

12. A method for software testing including the steps of:

    (a) Maintaining a data structure indicating the number of times conditional expressions and sub-expressions in conditional statements evaluate to *true/false*;

    (b) Upon reaching a conditional statement, evaluating the conditional expression and sub-expression and immediately updating the data structure;

    (c) Reporting conditional diversities computed using the counts from the data structure as a distance between the even distribution and the actual distribution of counts;

    (d) Maintaining a path trace containing execution paths represented as locations of the conditional expressions in the code coupled with the resulting Boolean values of the conditional expressions;

    (e) Upon reaching a conditional statement, evaluating the conditional expression and immediately updating the path trace;

    (f) Reporting path diversity computed using the path trace; and

(g) Reporting data diversity computed as an average of individual diversities, calculated as a percentage of test suites that have distinct conditional diversities for a conditional expression.

13. A test generation method as in claim 12 where the steps (a)-(g) apply to program executions where the Boolean values, that resulted from complete evaluation of the conditional expressions, might be substituted with different Boolean values.

14. A method as in claim 12 wherein steps (c) and (f) further include prioritization of diversities by sorting the diversities according to worse diversity order, and limiting their number in the diversity report to the top few worst diversities.

15. A method of software testing including the steps of:

(a) Automatic parsing of a computer program to locate conditional statements and conditional expressions within the conditional statements;

(b) Automatic insertion of instrumentation code around conditional expressions and sub-expressions;

(c) Execution of an instrumented program, including the step of generating conditional distribution data for each conditional expression and sub-expression, and including the step of generating a path trace;

(d) Computing conditional, data and path diversities from the distribution data and the path trace; and

(e) Reporting conditional, data and path diversities in an immediate, audible manner.

16. Software testing apparatus comprising of an instrumenter that accepts source files of different languages as input, parses the files, inserts instrument code at the conditional expression in conditional statements to provide instrumented source files; an executor that executes the program(s) containing the instrumented files, and generates conditional distribution and a path trace in response to the inserted instrument code; a conditional diversity calculator, which calculates conditional diversities from the conditional distribution and reports them; a data diversity calculator, which calculates data diversities from conditional diversities and reports them; and a path diversity calculator, which calculates path diversity from the path trace and reports them.

17. Apparatus as in claim 16 wherein the instrumenter inserts instrumentation code to automatically generate Boolean values and substitute these generated values for the actual values that result from complete evaluation of a conditional expression.

18. A storage medium comprising of means for storing an instrumented source files defining conditional statements; a means for storing a data structure indicating the *true/false* counts for the values of conditional expressions, and means for storing path traces indicating the values of the conditional expressions along the paths; and a means for storing a further data structure indicating the top worst condition, data and path diversities.

19. A storage medium as in claim 18 wherein the means for storing an instrumented executable program(s) includes means for storing a function that updates the conditional expression and sub-expression true/false counts, updates the path trace as the instrumented program(s) run and encounter conditional statements, and generates test values for the conditional expressions as the instrumented program(s) run and encounter conditional statements.